COPYRIGHT 1983 R.K.BENNETT

T INTRO

T DOSMAN

T NEWCMDS

T FEATURES

T DUPRES

T MAP

T A

T B

T C

T D

T E

T F

T G

T H

T I

T J

T K

T L

T M

T N

T O

T P

T Q

T R

T BUGS

T TIPS

*** END OF DEMO ***

INTRO - INTRODUCTION

! TYPE *RETURN* FOR MORE

journey through ATARI DOS 2.0S, as modified by Richard K. Bennett.

In this tutorial, the original DOS 2.0S will be referred to as "DOS-2.0", and the modified version as "DOS-MOD".

You will learn more about the capabilities, bugs, and flaws of DOS. The bugs have been fixed, some of the flaws removed, and some great new features and commands have been added.

NOTE: DOS-MOD fits in the same memory space as DOS-2.0!

! TYPE *RETURN* FOR MORE

DON'T BE INTIMIDATED

This tutorial was carefully designed for a wide range of users, from the neophyte to the software engineer.

If you are closer to the former than the latter, some of the material may not be clear on the first reading. Furthermore, it may be some time before you're ready to use all of the features available in DOS-MOD.

Therefore, if you find some parts of this tutorial difficult, it would be best if you skip over them on the first reading. Pick up what you can. You can go back later, when you are ready for more. For the present, enjoy the full screen use, the 1-line commands, the freedom from bugs, etc.

! TYPE *RETURN* FOR MORE

DEFINITIONS AND CONVENTIONS

BOOT: The process of loading DOS from disk on power-up.

BREAK: Break key. Normally used to abort an operation. Should not be used while DOS is writing a disk file.

CTRL: Control key. Ex: *CTRL*-1: hold *CTRL* key down & type "1".

HEX: Hexadecimal notation. Hex numbers are often indicated by "$". Ex: $80.

POWER-UP: The process of turning on the computer.

! TYPE *RETURN* FOR MORE

RAM: Random Access Memory, the

RAM: Random Access Memory, the memory used to store and run programs & data.

ROM: Read-Only Memory, used to hold the OS operating system.

RETURN: The carriage return key.

NB: A *RETURN* must be given at the end of each line you type before DOS will accept it.

! TYPE *RETURN* FOR MORE

TIPS, TRICKS, NEW COMMANDS & FEATURES

Included here are tips and tricks, some of which are in the DOS manual, but might be missed, & some are new. However, the emphasis is on the new commands and capabilities of DOS-MOD.

STRUCTURE OF THIS TUTORIAL

This tutorial consists of a set of "T" tutorial files, each covering one topic or command. These files are executed in sequence by another tutorial file, "DEMO" -- the one you probably started with.

The DEMO file, or any individual file, can be called by "T tutorial-name".

! TYPE *RETURN* FOR MORE

HOW TO SEE THE LIST OF TUTORIAL FILES

You can get the list of the tutorial files by typing "A *". The list is long: so be ready to use *CTRL*-1 to hold the screen.

! Type "A *" or *RETURN*.

! TYPE *RETURN* FOR MORE

USE OF "!" IN THESE TUTORIAL FILES

In these tutorial files, the screen is stopped by a "!" line, as you may have noticed. At this point you can type:

RETURN: to continue the demo, or
Any command: for DOS to execute, or
CTRL-3: to abort the tutorial file you are in, or
BREAK: to abort the whole tutorial.

Sometimes the "!" line suggests a command for you to type, but when it doesn't, the "!" line is just used to

to save space on the screen, the "!" line is overwritten when you type *RETURN*. So be aware: if you type a command in this case, the last line of output from your command will be lost.

! TYPE *RETURN* FOR MORE

TUTORIAL FILES FOR MENU COMMANDS

Each of the menu commands (A-R) is covered by a tutorial file of the same name - that is, to get the tutorial for command "A", type "T A".

The format of each command is given at the beginning of its file. For example

A [filespec][,listfile]

gives the format for the "A" Directory command. "Filespec" and "listfile" are defined in the DOS-2.0 manual. The default of "filespec" is "D1:*.*". Lower-case words indicate user input. Square brackets "[]" indicate optional information. Vertical bar "|" is used to indicate a choice between items: [/A|N|Q|P] = /A or /N or /Q or /P.

! TYPE *RETURN* FOR MORE

SKIPPING THROUGH THE DEMO FILE

As previously noted, the DEMO file calls a number of other files, including the INTRO file you are now in. If you want to skip ahead to the next file in the DEMO file, you can use *CTRL*-3 after a "!" line, to abort the file you are in, as covered above.

To facilitate skipping files, a "!" line has been placed at the start of each of the tutorial files, so you can move quickly ahead in the DEMO file, with a series of *CTRL*-3 keystrokes.

Later, after you have been through the whole DEMO tutorial, you can access any tutorial file individually (by "T filename") without going through the whole DEMO tutorial sequence.

! TYPE *RETURN* FOR MORE

THE TUTORIAL FILES REQUIRE OPEN FILES

Four open files are needed for this tutorial and DOS-MOD, as delivered, is configured to meet this requirement.

If you happen to get ERROR #161 (Too Many Open Files) you may have called a tutorial file within a tutorial file.

For a discussion of open files, see TIPS #7 in the TIPS tutorial, which follows at the end of this DEMO.

! TYPE *RETURN* FOR MORE

EXAMPLES OF COMMAND EXECUTION

In this tutorial, some commands are executed to show you their effects.

You can distinguish these "live" commands from tutorial text by the fact that live commands are left-justified, whereas text examples are indented by at least one space.

Also, you will have the opportunity to try executing commands yourself. When a line starts with "!", as described above, the screen scrolling stops and you can type in one command. Sometimes the text will suggest a particular command but you always have the choice of typing a command or just *RETURN*.

! TYPE *RETURN* FOR MORE

OSB-ROM or OSA-ROM?

The original Atari Operating System, OSA had some bugs, fixed in the OSB version released early in 1982.

Some DOS-MOD features labeled "OSB-ROM only" could not be included for OSA ROM, due to memory limitations resulting from the OSA bugs.

When you created DOS-MOD, a message on the screen told you which OS ROM you have. You can also tell your OS ROM by the suffix (A or B) on the MOD number in the heading, when you boot DOS-MOD.

! TYPE *RETURN* FOR MORE

In case you don't remember which OS you have, there is a simple way to find out right now, using the new "R" (Read Memory) command, which is only available for the OSB ROM. If it works, then you have OSB; else: OSA.

To tell if it works, after the R command line, you will see a line, which just happens to give the MOD number. Otherwise, the line after the R command line will just be "X". (More about the R command later.)

! TYPE *RETURN* FOR MORE

R IF48
X

If you do not have DSB, you may wish to upgrade your system. Contact your Atari dealer for information.

! TYPE *RETURN* FOR MORE

AUTORUN

If you have an 850 and use its RS-232 ports, rename AUTORUN to activate it:

E AUTORUN.MOD,AUTORUN.SYS

Then, on Boot, it will automatically load the RS-232 handler, if your 850 is on.

AUTORUN.MOD is the same as the DOS-2.0 AUTORUN.SYS, except it is tailored for DOS-MOD and a bug is fixed (BUGS #10).

NOTE: Don't rename the AUTORUN file now. The N tutorial will tell why.

! TYPE *RETURN* FOR MORE

MONEY-BACK GUARANTEE

We're serious about our money-back guarantee. We want to be sure you like this product before you decide to keep it.

If you decide to return it, just mail it back within 30 days and we will refund your money.

Of course, if you return it, you are honor bound to delete any files you have copied, including copies of DOS-MOD which you may have written to other disks.

! TYPE *RETURN* FOR MORE

COPYRIGHT

DOS-MOD, the tutorial, and the instruction booklet are copyrighted. We request that you honor this copyright.

Copy the DOS-MOD files as much as you like for your own use, but ask your friends to buy DOS-MOD for their use.

Thank you. We hope you enjoy this tutorial and derive much benefit from the DOS-MOD product.

***** END OF INTRO *****

DOSMAN: DOS II USERS MANUAL NOTES
! TYPE *RETURN* FOR MORE

1. READ YOUR DOS USERS MANUAL

You should have already read your DOS II Users Manual. It contains a good description of DOS 2.0S. Although we will review some of this material, no attempt is made to cover all of it.

2. TERMINOLOGY

The DOS manual uses the term "menu option" for what is really a "command", the term that is used in this tutorial.

! TYPE *RETURN* FOR MORE

3. FILENAMES

The manual states that an extension is three characters long. It should instead say that three characters is the MAXIMUM length.

! TYPE *RETURN* FOR MORE

4. WILDCARDS

The wild card "*" is not as general as the manual would lead you to believe.

Actually, "*" can only be used at the end of the file name or extension. If a "*" appears at the beginning or in the middle of one of these parts, any characters to the right of the "*" in the part are ignored.

For example, "A *MO" is equivalent to "A *". Since there is only one file on this disk ending with "MO" (DEMO) you would expect that "A *MO" would only list this file. However, you will find that this command will list all files on the disk with a blank extension!

! Try typing "A *MO". Or type *RETURN*

! TYPE *RETURN* FOR MORE

The wild card "?" is supposed to substitute for a single character. It works correctly when used at the beginning or in the middle of a name. But at the end of a name, it does not have the correct effect.

for example, you might expect that "A D?????" would list only 6-character file names starting with "D", but you will find that it lists all files of up to 6 characters, starting with "D".

! Try typing "A D?????". Or type *RETURN*

! TYPE *RETURN* FOR MORE

*** END OF DOS MANUAL NOTES ***

NEWCMDS: NEW COMMANDS
! TYPE *RETURN* FOR MORE

There are three new commands:

"P" Run Program: Permits the running of a program with a single letter.

"Q" Command File: Executes a set of commands from a file.

"R" Read/Store: (OSB-ROM only) Reads, displays, and stores bytes of data from memory.

Each of these commands is described in a tutorial file of the same name.

*** END OF NEWCMDS ***

#
FEATURES FILE
! TYPE *RETURN* FOR MORE

1. ONE-LINE COMMANDS

One of the new features is the option of typing a command on one line, instead of using three lines, as required by DOS-2.0.

To enter a one-line command, type the command letter, followed by one space, and then by the parameters. In this case, the prompt line which lists the parameters required is not displayed.

As an example, the command to get a directory list of .SYS files is shown, first in the only form available in DOS-2.0, which takes three lines, and then in the new one-line form you may use in DOS-MOD.

! TYPE *RETURN* FOR MORE

```
A
*.SYS
```

See the 3 lines in this, the only form available in DOS-2.0. Now look at the 1-line form you can use in DOS-MOD:

! TYPE *RETURN* FOR MORE

```
A *.SYS
```

Isn't that neater?

! TYPE *RETURN* FOR MORE

You can still use the old form, if you prefer, in order to get the prompt for the parameters.

In the 1-line command form, the prompt is not displayed, since the parameters are given in the command line.

If you would like to try the new way:

! Type "A *.SYS" or *RETURN*

Now if you want to try the old way, *first*, just type "A". You will then get the prompt message from DOS, which specifies the parameters you need to give. Then type "*.SYS".

! Type "A" or *RETURN*

! TYPE *RETURN* FOR MORE

2. DUP-RESIDENT FEATURE (OSB-ROM only)

This feature, if invoked, extends the memory area reserved for DOS, so that all of DOS can remain in memory.

The advantage of using this feature is the elimination of the reading and writing the MEM.SAV file, which takes about 30 seconds and has aggravated many users.

The disadvantage of using this feature is the loss of 5513 bytes of memory that the MEM.SAV facility makes available for program use, by allowing the sharing of memory between part of DOS (called DUP) and user programs.

This feature is fully discussed in the next tutorial file (DUPRES).

! TYPE *RETURN* FOR MORE

3. "READ DUP.SYS" MESSAGE
(OSB-ROM only)

when DUP.SYS is read into memory. This helps you understand what is happening while the disk is being accessed.

4. MEM.SAV READ/WRITE MESSAGES

Messages are written on the screen when MEM.SAV is written or read, for the same reason as for #3.

5. MENU COMPRESSED

The menu is compressed, allowing more room on the screen for workspace and history.

! TYPE *RETURN* FOR MORE

6. SCREEN-CLEARING MINIMIZED

Clearing the screen and re-displaying the menu have been minimized, thus leaving the history on the screen.

7. SCREEN CONSERVATION

The number of lines on the screen used by commands has been reduced, thus allowing more history to be displayed on the screen at a given time.

For example, queries to the user and the user's response are normally handled on a single line in DOS-MOD, whereas two lines are required by DOS-2.0.

! TYPE *RETURN* FOR MORE

8. NEW MENU-SELECTION CHARACTER

In DOS-2.0, *RETURN* is used to get a new menu display. Since *RETURN* is so frequently used, it is common to hit it in error. In DOS-2.0, this wipes out the history on the screen.

In DOS-MOD, the keystroke to select a new menu display is "0", to avoid invoking this action unintentionally.

! TYPE *RETURN* FOR MORE

9. NEW MENU DOES NOT CLEAR SCREEN

In DOS-2.0, when a new menu is displayed, the screen is cleared and the history on the screen is lost.

In DOS-MOD, the screen is not cleared, thus leaving most of the history on the screen.

! TYPE *RETURN* FOR MORE

(OSB-ROM only)

In DOS-2.0, a BRK instruction causes the system to "crash" (get stuck!).

DOS-MOD traps the 6502 BRK instruction and returns control to the DUP Menu, and hence to you.

11. LOGGING OF FILE OPERATIONS

All files copied, deleted, or renamed are logged on the screen to give the user a record of the operations.

! TYPE *RETURN* FOR MORE

12. DUP-AREA CHECK-SUM

When DUP exits, a multibyte checksum of the DUP-area of memory is computed and saved. Before DUP is re-entered, the checksum is recomputed & compared with the value saved. A difference indicates that the DUP-area has been used by a program and therefore DUP has been corrupted & must be reloaded.

If the MEM.SAV facility is "ON", the program or data in the DUP-area will be written to the MEM.SAV file before DUP is reloaded.

This checksum provides a positive method of determining both the integrity of DUP and the use of the DUP-area by a program.

! TYPE *RETURN* FOR MORE

13. WILDCARD IN THE DISK DRIVE NUMBER

A wildcard (*) may be used as the disk drive number for the Directory "A" and Delete "D" commands.

! TYPE *RETURN* FOR MORE

14. BETTER WAY TO SPECIFY COPY BUFFER

In DOS-2.0, whether an internal buffer or program memory is to be used by the Copy "C" command is determined by a query to the user.

In DOS-MOD, this choice is controlled instead by a option on the filename (/P: Program memory).

This use of an option, rather than a query response, is an easier method for the user to control which space is to be used for copy operations.

See the "C" tutorial for more info.

! TYPE *RETURN* FOR MORE

15. MEMORY USED FOR COPY IS OPTIMIZED

In DOS-MOD, the amount of memory used is optimized to obtain the maximum copy speed with minumum memory.

In contrast, DOS-2.0 uses all of program memory, which destroys much more memory than necessary. Also, if you want to display just the first few lines of a file, you must wait until all of memory is filled before Copy will display any of the file.

16. QUERY FLAG FOR COPY COMMANDS

Copy "C", Duplicate-File "D", & Rename "E" commands are now provided with a query option (/Q), which allows the user to control whether or not to copy individual files in a wildcard series.

! TYPE *RETURN* FOR MORE

17. INADVERTENT FILE-DELETION AVERTED

In DOS-2.0, on a Copy "C" command, if the destination file already exists, the old file will be deleted by the copy process, without warning.

In DOS-MOD, the old file is protected from inadvertent deletion by a query (unless the new no-query option /N is used). If the user answers "Y" (Yes), the copy will take place. Otherwise, the operation on that file will not be performed.

This feature is covered more completely in the "C" tutorial file.

! TYPE *RETURN* FOR MORE

18. WILDCARD USE IN COPY TO SAME DISK

In DOS-2.0, wildcards can only be used when copying files from one disk-drive unit to a different disk-drive unit.

In DOS-MOD, wildcards can be used when copying files to any drive, including the same drive.

19. FILE CONCATENATION

In DOS-2.0, if wildcards are used in the source filespec, they must also be used in the destination filespec.

In DOS-MOD, you have a choice. You can use wildcards in the source filespec only, and thus concatenate or append a

destination file by a single command.

! TYPE *RETURN* FOR MORE

20. M COMMAND CAN RUN A LOADED FILE

The Run "M" command can now be used to run the last binary file loaded by the "L" command. (See the M tutorial file)

21. MEM.SAV FACILITY FIXED

MEM.SAV has been completely redesigned to work correctly and appropriately.

This facility is fully described in the N tutorial file for the N command.

22. ILLEGAL COMMAND LINES IGNORED

If the first character of a command line is not a legal command character (i.e., in the Menu list), then the line is just ignored.

! TYPE *RETURN* FOR MORE

23. FULL USE OF SCREEN

Notice, you now have the whole screen to view. In DOS-2.0, you only had a few lines in the bottom portion. The rest of the screen was normally reserved for the Menu, which is not needed all the time!

24. IMPROVED COMMAND PROMPTS

The prompt line which is displayed when you don't give parameters on the command line, tells what parameters are required. In DOS-MOD, the prompt lines have been improved & simplified and now include the "/" Options.

*** END OF FEATURES ***

DUPRES FILE: DUP-RESIDENT OPTION

! TYPE *RETURN* FOR MORE

OSB-ROM ONLY

This option is only available if you have the OSB-ROM. Review the INTRO tutorial file for more information about this question.

STUCTURE OF DOS

To understand the DUP-Resident option,

structure of DOS.

DOS consists of two parts:

1. DOS.SYS
2. DUP.SYS

! TYPE *RETURN* FOR MORE

DOS.SYS: MEMORY-RESIDENT PART

DOS.SYS is memory "resident". It is booted into RAM memory on power-up and must remain undisturbed in memory thereafter.

DUP.SYS: NON-MEMORY-RESIDENT PART

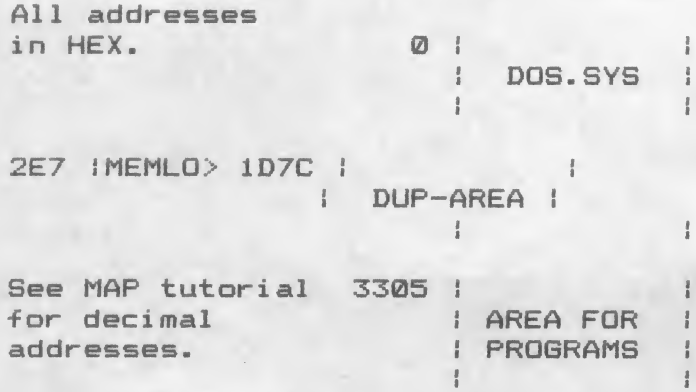
DUP.SYS handles the Menu commands and needs to be in RAM only when you are using them. When you run a program, you don't need the Menu commands, so DUP is not needed and this "DUP-area" of memory can be used by programs, including BASIC and ASSEMBLER.

NOTE: "DUP" here stands for "Disk Utility Program" and is not the abbreviation of "Duplicate" used with the "J" and "O" commands.

! TYPE *RETURN* FOR MORE

MEMLO: LOWEST LOCATION IN MEMORY

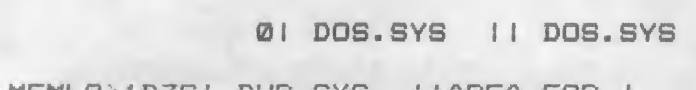
MEMLO is a 2-byte pointer maintained by DOS (at 2E7) to establish the first location in RAM that a program may use for storage. It normally points to the beginning of the DUP-area:

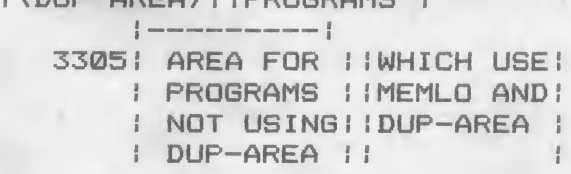


! TYPE *RETURN* FOR MORE

SHARING THE DUP-AREA

Sharing the DUP-area gives programs more memory to use, but requires swapping memory between RAM & disk.





A program, like BASIC, that uses MEMLO to locate its buffer, will overwrite DUP in the DUP-area. Before returning to the Menu, DUP must be restored from the DUP.SYS file, which takes time.

! TYPE *RETURN* FOR MORE

SWAPPING PROGRAM DATA IN THE DUP-AREA

If a program has used the DUP-area, the data in the DUP-area must be saved to disk before DUP.SYS is read back into the DUP-area. Otherwise, that data will be lost.

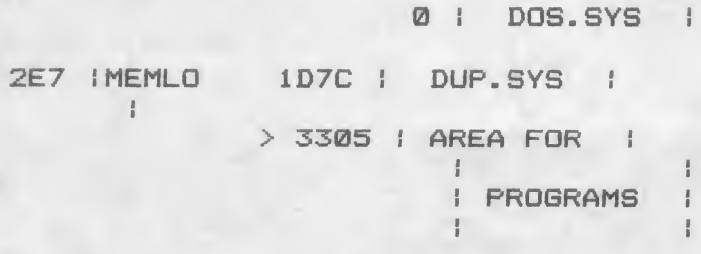
This process of saving program data on disk and restoring it back in RAM is called "swapping" and is performed by the "N" MEM.SAV command, described in the N tutorial file. This process takes even longer than restoring DUP from the DUP.SYS file.

The purpose of the DUP-Resident option is to avoid this time-consuming and sometimes troublesome swapping process.

! TYPE *RETURN* FOR MORE

CHANGE MEMLO TO POINT BEYOND DUP-AREA

The method used by the DUP-Resident option is simple: MEMLO is changed to point beyond the DUP-area:



With this option, a program (like BASIC) which uses MEMLO to locate its buffer will not overwrite DUP, thus obviating the need to swap and restore the DUP-area.

! TYPE *RETURN* FOR MORE

PENALTY FOR USING DUP-RESIDENT OPTION

The price you pay for selecting the

5513 bytes of memory available to programs. This means smaller buffers in BASIC for your program and in word-processors for your text, etc.

But the price might be worth it, if you frequently go back & forth between the Menu commands and programs.

! TYPE *RETURN* FOR MORE

CHECK-SUM PROTECTION OF DUP

As mentioned in FEATURES-12, DUP is protected by a check-sum generated when leaving the Menu commands and re-computed when returning. If a change in DUP is detected, DUP will be restored from the DUP.SYS file.

Even if you select the DUP-Resident option, a program could still corrupt DUP, either due to a bug or due to the fact that some programs do not use MEMLO, as they should.

If DUP is thus corrupted, it will be automatically restored from DUP.SYS. You'll see the message "READ DUP.SYS" on your screen, telling you that old faithful DOS-MOD is on top of the situation.

! TYPE *RETURN* FOR MORE

SETTING THE DUP-RESIDENT BYTE

To enable the DUP-Resident feature:

```
POKE 5455,128    [BASIC]
C 154F<80        [ASSEMBLER "BUG"]
```

To disable the DUP-Resident feature,

```
POKE 5455,0      [BASIC]
C 154F<0         [ASSEMBLER "BUG"]
```

After the POKE, the DOS files must be saved by the "H" command and the system re-booted.

*** END OF DUPRES ***

MEMORY MAP

! TYPE *RETURN* FOR MORE

DEC	HEX	
0	0	6502 Page-0 for OS
128	80	6502 Page-0 for Cartridges
256	100	6502 Stack

```
 1152  480 RAM for Cartridges
 1536  600 Free RAM
 1792  700 FMS: File Management System
 5440 1540 Mini-DUP
 6780 1A7C File Buffers for FMS
 7548 1D7C DUP: Disk Utility Program
13061 3305 Free RAM
40960 A000 ROM for Cartridges
49152 C000 Not used
53248 D000 I/O ports
55296 D800 OS ROM

Note: DOS.SYS file has FMS & Mini-DUP
      DUP.SYS file has DUP

! TYPE RETURN FOR MORE

Useful individual locations

 5444 1544 MEM.SAV-File-Valid-Data Flg
 5445 1545 MEM.SAV-Feature-ON Flag
 5446 1546 Use-Program-Memory Flag
 5455 154F DUP-Resident-Option Flag
 6275 1883 Left Margin (OSA-ROM)
 6390 18F6 Left Margin (OSB-ROM)

POKE 128 ($80) to set a flag; 0 to
clear it.  The MEM.SAV flags are
discussed in the N tutorial, the Use-
Program-Memory Flag in C, and the DUP-
Resident-Option Flag in DUPRES.

POKE the column number to set the Left
Margin.  Then re-boot or SYSTEM-RESET.
(Note: 1st column number = 0.)

! TYPE RETURN FOR MORE

LEFT MARGIN

In DOS-2.0 and OS, the Left Margin is
set at 2.

In DOS-MOD, the Left Margin is 0.

The advantage of 2 is that some TV's
cut off the left margin. The advantage
of 0 is the use of the full 40 columns
of the screen.

NOTE:  These tutorial files assume the
       Left Margin to be 0.  If you
       change it from 0, lines will
       overflow on the screen.

      *** END OF MEMORY MAP ***


"A" - DIRECTORY LISTING
! TYPE RETURN FOR MORE
```

EXAMPLE

```
A *.OBJ,DIRLIST.LST
```

The "A" command is well described in the DOS manual. Note that the default listfile is (E:), the Screen Editor.

In DOS-MOD, Bug-9 is fixed (see the BUGS file) and the ability is added to use the "*" wildcard as the disk device unit number: "D*:".

If you have more than one disk drive:

! Try "A D*:" or type *RETURN*

*** END OF A ***

"B" - RUN CARTRIDGE

! TYPE *RETURN* FOR MORE

```
B
```

No change from DOS-2.0.

*** END OF B ***

"C" - COPY FILES

! TYPE *RETURN* FOR MORE

```
C [from-file],[to-file][/A|Q|N|P]
```

The COPY command in DOS-MOD performs the same function as in DOS-2.0, but there are major improvements.

One improvement allows the user to omit the to-filename, if it is the same as the from-filename. For example, in DOS-MOD, both:

```
C ANYFILE.TXT,D2:
C ANYFILE.TXT,D2:ANYFILE.TXT
```

will copy ANYFILE.TXT from Disk-1 to Disk-2. (DOS-2.0 permits only the long form, which takes more typing & could result in an undetected error if the user mispells the to-filename.)

! TYPE *RETURN* FOR MORE

WILDCARDS

Wildcards may be used in both the "from" and "to" filespecs to copy a set of files, changing their names at the same time. For example,

```
C *.TXT,*.ASC
```

will copy all ".TXT" files as ".ASC" files.

Note: DOS-MOD allows wildcards when copying files from one drive to any other drive, including the same drive, whereas DOS-2.0 disallows wildcards when copying files to the same drive.

DOS-MOD also permits wildcards in just the to-file, which can simplify single-file copying (see TIPS #5).

! TYPE *RETURN* FOR MORE

A new feature of DOS-MOD now allows a set of wildcard files to be copied to a single file. (This is described below in "/A" and in "CONCATENATION".)

OPTIONS

In addition to the /A option available in DOS-2.0, DOS-MOD provides three new options:

```
/A - Append
/N - No query for delete
/Q - Query for copy
/P - Program memory for buffer
```

Only one option at a time may be used, except that the /P option may be used with any one of the others.

! TYPE *RETURN* FOR MORE

"/A" - APPEND OPTION

This option is the same as in DOS-2.0, except that DOS-MOD now allows a number of files to be appended with one command line, using wild cards.

For example, several text files could be appended to a given file by:

```
C *.TXT,COLLECT.TXT/A
```

The COLLECT.TXT file must pre-exist or an error is reported. (To append to a non-existent file, see CONCATENATION, below.)

The ".TXT" files will be appended in the order that they appear in the directory. Thus, this form can only be used if the order is not important.

! TYPE *RETURN* FOR MORE

CONCATENATION

A feature of DOS.MOD will concatenate a set of wildcard files to a new file. No option is needed for this case. It is handled automatically. For example:

```
C *.TXT,COLLECT.TXT
```

will create COLLECT.TXT, if it does not exist. If COLLECT.TXT already exists, then DOS-MOD will ask:

```
DELETE D1:COLLECT.TXT?
```

A "Y" response will cause the existing file to be deleted and all .TXT files to be appended to a new COLLECT.TXT file. Any other response (normally just a *RETURN*) will treat the command as though it had had an "/A" option.

! TYPE *RETURN* FOR MORE

CONCATENATION ILLUSTRATED

To illustrate concatenation, first we create three test files by copying from tutorial files:

```
C B,B.TST/N
C F,F.TST/N
C G,G.TST/N
```

Incidentally, the /N option (described below) was used to force the copy, in case the ".TST" files had already been created. Otherwise, a "DELETE?" query would have held up the copy process.

! TYPE *RETURN* FOR MORE

Now we concatenate these files:

```
C *.TST,COLLECT.TST/N
```

Again, the /N option was used, in case COLLECT.TST already existed.

NOTE: Only the first line:

```
C *.TST,COLLECT.TST/N
```

was in this command file. The other 3 lines came from DOS-MOD.

Now you can look at the concatenated file by copying it to the screen (E:). (Be sure you are ready with *CTRL*-1.)

! Type "C COLLECT.TST,E:" or *RETURN*

! TYPE *RETURN* FOR MORE

/N - NO QUERY FOR DELETE OPTION

This option suppresses a feature of DOS-MOD, one that protects against an inadvertent deletion of the to-file.

If the to-file exists, it will be deleted by the copy process. In DOS-2.0, this deletion occurs without warning! In DOS-MOD, however, the user is queried about the deletion.

Ex: If to-file exists, DOS-MOD asks:

DELETE D2:ANYFILE.TXT?

A "Y" response causes the copy to take place. Otherwise the copy is skipped.

If /N had been used, the delete & copy would have occurred without the query.

! TYPE *RETURN* FOR MORE

"/Q" - QUERY OPTION

This new option of DOS-MOD queries the user before each wildcard copy.

There are two cases: one where the to-file exists and the second where it does not exist.

! TYPE *RETURN* FOR MORE

If the to-file exists, the query asks if it may be deleted, as in the above example. A "Y" response permits the deletion, and hence the copy. Any other response skips the copy.

If the to-file does not exist, the query asks (in the above example):

COPY - D1:ANYFILE.TXT?

A "Y" response enables the copy. Any other response skips the copy.

Thus, the user can select which files of a wildcard series will be copied.

Also, a record is left of the from-files that were copied and the to-files that were deleted. (A deleted to-file implies a copied from-file.)

! TYPE *RETURN* FOR MORE

To illustrate /Q, let's first make sure we have no ".TMP" files.

We won't use the /N option this time, so you will have to answer "Y" to delete the files.

D *.TMP

! TYPE *RETURN* FOR MORE

Now we create one ".TMP" file:

C B,B.TMP

Now we copy the ".TST" files to ".TMP"

C *.TST,*.TMP/N

Since B.TMP existed before this copy, it was deleted by this copy and thus it appears in the list as a "DELETE", rather than a "COPY".

! TYPE *RETURN* FOR MORE

"/P" - PROGRAM-MEMORY OPTION

This option specifies that program memory, rather than an internal 256-byte buffer, is to be used to buffer the copy operation. This is a new method of specifying what memory area is to be used as the buffer.

In DOS-2.0, you are asked if it is OK to use program memory. If you do not consent, then the internal buffer is used. This query is repeated on every copy until you say "Y". When you do, DOS-2.0 then uses and destroys all of program memory.

In DOS-MOD, the choice of buffer is controlled by the /P option, instead of by query. "/P" indicates that the program memory is to be used.

! TYPE *RETURN* FOR MORE

The use of program memory, which is much larger than the internal buffer, results in a faster copy. But the use of program memory destroys any data it might have contained.

The amount of program memory used by DOS-MOD is 3104 ($C20) bytes. It was optimized to give the maximum copy speed with a minimum of memory. Furthermore, DOS-MOD uses only the top of program memory, so as not to affect programs in lower memory.

However, a cartridge, such as BASIC, uses all program memory, so the use of /P may corrupt your BASIC program. To prevent the use of a corrupted program DOS signals BASIC to re-initialize and wipe out your program. Don't use /P if you want to retain your BASIC program.

! TYPE *RETURN* FOR MORE

The /P option can be used with any of the other options. The order does not matter.

Also, the /P option need only be used once while you are in the DUP Menu. After the first use, subsequent Copy operations will use program memory.

USE-PROGRAM-MEMORY FLAG

If you always want to use program memory for the copy buffer, set the flag at 5446 ($1546) to 128 ($80). (Zero clears it.) Then, the /P option is not needed. Save DOS (H command) to make this selection permanent.

*** END OF C ***

$
"D" - DELETE COMMAND
! TYPE *RETURN* FOR MORE

D [filespec][/N]

This command is essentially the same as in DOS-2.0, except that the "*" wildcard may be used for the disk drive unit number (cf: "A" Directory command).

To illustrate the "D" command, let's delete the ".TST" files we created in the "C" tutorial.

We won't use the /N option so as not to delete any ".TST" files you might have on your disk. So you will have to respond with "Y" for each file, to delete it.

! TYPE *RETURN* FOR MORE

D *.TST

*** END OF D ***

"E" - RENAME FILE
! TYPE *RETURN* FOR MORE

E D#:old-name,new-name[/N!Q]

Where "#" is the disk drive unit

be omitted.) Note that the device designation (D#:) applies to both the old and new file names. Wildcards may be used in both the old and new filenames.

DOS-MOD has fixed Bug #2 (see the BUGS file), so that two files of the same name cannot be created, as would occur in DOS-2.0 if a file were renamed to an existing filename.

DOS-MOD has also added the options: /N and /Q.

! TYPE *RETURN* FOR MORE

EXAMPLES

1. E ANYFILE.TXT,NEWNAME.TXT
2. E D2:*.TXT,*.ASC

If the option /N is not used and a "new-name" file already exists, the user will be asked (Example 1):

DELETE D1:NEWNAME.TXT?

A "Y" reply causes the existing file NEWNAME.TXT to be deleted and the file ANYFILE.TXT to be renamed NEWNAME.TXT. Any other reply skips the renaming. If wildcards are used, the next file will then be handled.

! TYPE *RETURN* FOR MORE

"/N" - NO-QUERY-FOR-DELETE OPTION

If you know ahead of time that you want the rename operation to be executed whether or not a "new-name" file exists, and don't want to be bothered answering dumb queries, use the /N option.

This option authorizes, in advance, the deletion of a "new-name" file, so that the rename operation may proceed without waiting for a query.

! TYPE *RETURN* FOR MORE

"/Q" - QUERY OPTION

This option queries the user before each file of a wildcard rename.

If the new-name file exists, the query asks if the new-file may be deleted, as in the above example. A "Y" reply permits the deletion and hence the rename.

If the new-file does not exist, the

RENAME D1:ANYFILE.TXT?

A "Y" reply enables the rename. Any other reply skips the file.

Thus the user can select which files of a wildcard series will be renamed.

! TYPE *RETURN* FOR MORE

*** END OF E ***

"F" - LOCK FILE

! TYPE *RETURN* FOR MORE

F [filespec]

No change from DOS-2.0.

*** END OF F ***

"G" - UNLOCK FILE

! TYPE *RETURN* FOR MORE

G [filespec]

No change from DOS-2.0.

*** END OF G ***

"H" - WRITE DOS FILES

! TYPE *RETURN* FOR MORE

H disk-drive-unit-#

Example:

H 1

No change from DOS-2.0.

*** END OF H ***

"I" - FORMAT DISK

! TYPE *RETURN* FOR MORE

I disk-drive-unit-#

Example:

I 1

No change from DOS-2.0.

*** END OF I ***

"J" - DUPLICATE DISK
! TYPE *RETURN* FOR MORE

J from-drive-#,to-drive-#

Example:

J 1,2

No change from DOS-2.0.

*** END OF J ***

K - BINARY SAVE
! TYPE *RETURN* FOR MORE

K filespec[/A!N],start,end
[,init][,run]

Example:

K D2:ANYFILE.SAV/A,5000,5FFF,,5000

! TYPE *RETURN* FOR MORE

/A - APPEND OPTION

This option is the same as in DOS-2.0. It causes the binary data addressed by the "start" and "end" locations to be appended to an existing file.

! TYPE *RETURN* FOR MORE

/N - NO-QUERY-FOR-DELETE OPTION

This option suppresses a feature of DOS-MOD, one that protects against an inadvertent deletion of the "save" file.

If the "save" file exists, it will be deleted by the "save" process. In DOS-2.0, this deletion occurs without

user is queried about the deletion.

The /N option suppresses the query.

This option is the same as for the C Copy and E Rename commands.

*** END OF K ***

"L" - BINARY LOAD
! TYPE *RETURN* FOR MORE

L filespec[/N!O]

Example:

L D2:ANYFILE.SAV/N

In addition to the /N option available in DOS-2.0, DOS-MOD provides a second option (/O) to allow suppressing the Run address, without suppressing the Init address.

/N - No Run or Init
/O - Only Init (no Run)

! TYPE *RETURN* FOR MORE

"/N" - No-Run-or-Init Option

This option has the same effect as it does in DOS-2.0, inhibiting both the Init and Run addresses.

! TYPE *RETURN* FOR MORE

"/O" - Only-Init Option

This new option inhibits only the Run function. The Init function remains active. This option is needed when the Init action is needed for the loading process to proceed.

*** END OF L ***

"M" - RUN AT ADDRESS
! TYPE *RETURN* FOR MORE

M [hex-address]

Example:

M 5C00

The only change from DOS-2.0 is the option of giving no address. In this case, the Run address from the last file loaded by the L command is used.

If you have occasion to Load a program with the /N option, which inhibits its running, this feature gives you a simple method of running it later.

*** END OF M ***

"N" - MEM.SAV FACILITY
! TYPE *RETURN* FOR MORE

N F!O!C

In DOS-MOD, the N command requires a parameter of a single letter to specify the action to be taken:

"F" - OFF: Turns the facility OFF
"O" - ON: Turns the facility ON
"C" - CLEAR: Clears the MEM.SAV file

Example:

N C

"clears" the MEM.SAV file, that is, the MEM.SAV file is marked as not containing valid data. A file so marked will not be loaded into RAM the next time you run a program.

! TYPE *RETURN* FOR MORE

The MEM.SAV facility and the N command which controls it serve the same purpose as in DOS-2.0.

In DOS-2.0, the N command creates the MEM.SAV file, which turns ON the MEM.SAV facility. Deleting the file turns the facility OFF.

In DOS-MOD, the MEM.SAV facility has been redesigned to correct bugs, and the N command now provides better control. Time-consuming file operations are not required.

! TYPE *RETURN* FOR MORE

MEM.SAV FACILITY OVERVIEW

The MEM.SAV feature confuses everyone.

Let's carefully review the MEM.SAV process, because it is important to

from your Atari computer.

The MEM.SAV feature is very valuable. It makes available an additional 5513 bytes of memory for your programs.

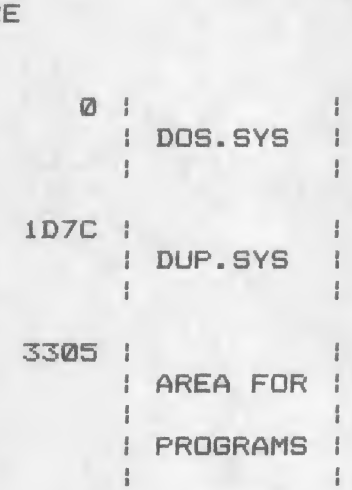
Where do these additional bytes come from? They are shared with part of DOS.

! TYPE *RETURN* FOR MORE

STRUCTURE OF DOS

DOS has 2 parts:

1. DOS.SYS
2. DUP.SYS



DOS.SYS must always remain in memory.

DUP.SYS need only be in memory while you are using Menu commands. This "DUP-area" can be shared with your programs.

! TYPE *RETURN* FOR MORE

SHARING MEMORY BY DUP.SYS AND PROGRAMS

The DUP-area can be shared by DUP.SYS and user programs in two ways.

First, by restoring DUP.SYS from disk when going to the DUP Menu from a program. In this case, the program's data in the DUP-area is lost.

Second, by swapping the program's data to and from disk (MEM.SAV file), when going between the DUP Menu and your program.

! TYPE *RETURN* FOR MORE

SWAPPING DUP-AREA MEMORY

If a program uses the DUP-area, and you want to be able to return to your program after going to the DUP Menu, then the data in the DUP-area must be saved before DUP.SYS is read in over it.

In this case, the program's data in the DUP-area is written to the MEM.SAV file when you go to the DUP-Menu and

return to your program.

This function is also performed when you load programs into the DUP-area or write binary files from it.

! TYPE *RETURN* FOR MORE

FACTORS CONTROLLING MEM.SAV ACTIONS

There are several factors that affect the action the MEM.SAV facility will take. These include:

Is the MEM.SAV facility ON?

Is the MEM.SAV file data valid?

Has a program written data in the DUP-area?

Has the data in the MEM.SAV file been loaded into the DUP-area?

Is a program about to be written from, or loaded to, the DUP-area?

Let us look at several cases involving these factors.

! TYPE *RETURN* FOR MORE

1. A PROGRAM WRITES IN THE DUP-AREA

The 1st case to be considered is when a program writes into the DUP-area.

The Atari BASIC cartridge is such a program and will be used as the example for this case. BASIC uses the DUP-area for storing the user program.

EXCEPTION: If the DUP-Resident option is enabled, the DUP-area is not used by programs like BASIC which correctly use MEMLO to determine where to begin their storage area. See the DUPRES tutorial for info about this option.

The action taken in Case-1 depends on several factors as will be seen below.

! TYPE *RETURN* FOR MORE

IF MEM.SAV FACILITY IS OFF

If MEM.SAV is OFF, when you go to the DUP Menu from BASIC (by typing "DOS"), your program in the DUP-area is not saved on disk. Consequently, your BASIC program will be destroyed when the DUP Menu program is restored to the DUP-area (from the DUP.SYS file).

If you had SAVE'd your BASIC program before you left BASIC and if you don't plan to return to your BASIC program from the Menu, then you have no problem: it is OK that the MEM.SAV facility was OFF.

However, if you want to find your program intact when you return to BASIC, then MEM.SAV should be ON.

! TYPE *RETURN* FOR MORE

IF MEM.SAV FACILITY IS ON

If MEM.SAV is ON when you go to the DUP Menu from BASIC, your program in the DUP-area is saved on disk in the MEM.SAV file, before DUP is restored from the DUP.SYS file.

IF MEM.SAV FILE HOLDS VALID DATA

In this case, where MEM.SAV was ON and your program in the DUP-area was written to the MEM.SAV file on disk, DOS-MOD takes note that the MEM.SAV file holds valid data.

When you return to BASIC from the Menu (by typing "B"), DOS-MOD loads the MEM.SAV file into the DUP-area.

Back in BASIC, your program is intact.

! TYPE *RETURN* FOR MORE

IF MEM.SAV FILE DATA NOT VALID

Suppose you decide that you don't want your BASIC program restored when going back to BASIC. Or maybe you are going to load and run another program, so that the data in the MEM.SAV file is no longer valid.

In this case, you can notify DOS-MOD to inhibit the reading and writing of the MEM.SAV file either by turning the MEM.SAV facility OFF:

```
N F
```

or by clearing the MEM.SAV file:

```
N C
```

which leaves the MEM.SAV facility ON.

! TYPE *RETURN* FOR MORE

WHY NOT LEAVE THE MEM.SAV FACILITY ON

Before continuing to the next case, let's answer the question: If the

leave it ON all the time?

The reason is simple: It takes a long time -- about 30 seconds -- to save and restore the DUP-area to and from the MEM.SAV disk file.

So when you don't need it, you'll want to turn OFF the MEM.SAV facility.

If you need the MEM.SAV facility ON, but want to stop it from saving and restoring invalid data, use CLEAR in the "N" MEM.SAV command, as discussed above. That will stop the wasted operations, but leave the facility ON.

! TYPE *RETURN* FOR MORE

2. LOADING A PROGRAM TO THE DUP-AREA

The 2nd case is where you load -- but do not run -- a binary program that overlaps the DUP-area.

PROGRAM, NOT DATA, IN THE DUP-AREA

Note that this case is different from Case-1.

In Case-1, the program did not itself overlap the DUP-area, but rather it stored data (your BASIC program, for example) in the DUP-area, when it ran.

In this case, Case-2, the program itself is loaded into the DUP-area (by the "L" command). This presents a special problem to DUP, since DUP gets overwritten in the process!

! TYPE *RETURN* FOR MORE

HOW DUP LOADS TO THE DUP-AREA

You might wonder how, when you are in the DUP Menu, you can use the "L" Load command to load a file to the DUP-area, where DUP itself resides.

The answer is that the loader is not in the DUP-area, but rather in the DOS (memory-resident) area. All DUP does for an "L" command is to get the file information from you and then transfer control to DOS to perform the load.

Of course, if the binary file you are loading contains data to be loaded in the DUP-area, it will destroy the DUP Menu program in the process. DOS takes care of this by reloading DUP.SYS into the DUP-area before returning to the DUP Menu.

IF MEM.SAV FILE HOLDS VALID DATA

If the MEM.SAV file holds valid data, then it will be loaded into the DUP-area before a binary program which overlaps the DUP-area is loaded.

The reason is that the MEM.SAV file may hold data in a different part of the DUP-area than that used by the binary file. If the MEM.SAV file were not loaded first, the data in the MEM.SAV file would be lost when the DUP-area is later written to the MEM.SAV file.

! TYPE *RETURN* FOR MORE

AFTER THE LOAD, BACK TO THE MENU

After the binary load is completed, we must get back to the DUP menu. (Remember, in this case, Case-2, the program is loaded -- but not run. So we have to go back to the Menu.)

To get back to the Menu, DUP must be restored to the DUP-area from the DUP.SYS file. In the process, the binary program just loaded is overwritten and destroyed. To avoid the loss of the program data in the DUP-area, it must first be saved in the MEM.SAV file.

Thus, the MEM.SAV facility must be ON before loading a binary file into the DUP-area, unless it is to be run as part of the load, at its Run address.

! TYPE *RETURN* FOR MORE

RECAP OF CASE-2

Loading -- but not running -- of a binary file, which overlaps the DUP-area, takes the following steps:

The MEM.SAV file is loaded into the DUP-area, if the MEM.SAV file holds valid data and the MEM.SAV facility is ON.

The binary file is loaded.

The DUP-area is saved to the MEM.SAV file, if the MEM.SAV facility is ON.

The DUP.SYS file is restored to the DUP-area.

The DUP Menu program is reentered.

! TYPE *RETURN* FOR MORE

3. RUNNING A PROGRAM IN THE DUP-AREA

This case is related to Case-2.

In Case-2, the program in the DUP-area is not run. In this case, Case-3, the program is run, using the Run-Address contained in the binary file.

This difference is important, since in this case, we do not have to return to the DUP Menu after the load. Thus, the DUP-area does not have to be saved to the MEM.SAV file and DUP does not have to be restored from the DUP.SYS file.

However, if the MEM.SAV file holds valid data, the DUP-area will still be restored from the MEM.SAV file before the binary file is loaded into the DUP-area.

! TYPE *RETURN* FOR MORE

RE-CAP OF CASE-3

The following steps are taken in Case-3, where a binary file that overlaps the DUP-area is loaded & run:

The DUP-area is restored from the MEM.SAV file, if the MEM.SAV file holds valid data.

The binary file is loaded.

The program is run.

NOTE: Case-3 is simpler than Case-2, because we do not return to the DUP Menu after the load, since the program is immediately run. Thus, the DUP Menu does not have to be restored or the program in the DUP-area saved.

! TYPE *RETURN* FOR MORE

4. RUN A PROGRAM NOT IN THE DUP-AREA

Before running a program, even if not in the DUP-area, the MEM.SAV file will be loaded to the DUP-area, if it holds valid data.

5. BINARY SAVE FROM THE DUP-AREA

This case is where the area to be written by a Binary Save "K" command includes any part of the DUP-area.

If the MEM.SAV file holds valid data, this file will be loaded to the DUP-area, before the binary file is

*** END OF CASES ***

! TYPE *RETURN* FOR MORE

MEM.SAV FACILITY: ON or OFF WHEN BOOT

DOS-MOD comes with the MEM.SAV facility OFF. If you want DOS-MOD ON when you boot your system, turn it ON by "N 0" and save DOS files by "H 1".

AUTOMATIC TURN-ON FOR RS-232 HANDLER

In DOS-2.0, the RS-232 handler is destroyed if the MEM.SAV file does not exist before booting your system.

In DOS-MOD, the RS-232 handler is saved by the automatic turn-ON of the MEM.SAV facility, when the handler is loaded. This all occurs if an ATARI 850 Interface Module is connected, and the AUTORUN.MOD file has been renamed AUTORUN.SYS. (See also "AUTORUN" in the INTRO tutorial file.)

! TYPE *RETURN* FOR MORE

HOW TO TURN ON MEM.SAV WHILE IN BASIC

If you find yourself in BASIC (or ASSEMBLER) with the MEM.SAV facility OFF, you can turn it ON by:

```
POKE 5445,128    [BASIC]
C 1545<80        [ASSEMBLER "BUG"]
```

(Or you can turn the facility OFF by POKE'ing a 0 in the same location.)

! TYPE *RETURN* FOR MORE

SIZE OF THE MEM.SAV FILE

The MEM.SAV file is 45 sectors long. Room for this file must exist on the disk in drive-1, to use the MEM.SAV facility.

NOTE: This tutorial is so large that there is not enough room on your "tutorial" disk for the MEM.SAV file. Wait until you have DOS-MOD on another disk before turning ON the MEM.SAV facility (or renaming AUTORUN.MOD).

*** END OF N ***

"O" - DUPLICATE FILE

O [filespec][/Q]

Example:

O ANYFILE.SAV

This command is used only if you have just one disk drive and want to copy a file from one diskette to another.

! TYPE *RETURN* FOR MORE

There are two changes from DOS-2.0:

1. The addition of the /Q option for Query on a wildcard series. (See the C tutorial.)

2. The requirement that program memory be used for the buffer, as it is for the Duplicate Disk "J" command.

   DOS-MOD asks if program memory may be used. "Y" allows the command to proceed. Otherwise, it is aborted.

   (DOS-2.0 allows the use of a small internal buffer, which results in excessive disk switching.)

*** END OF O ***

"P" - RUN PROGRAM

! TYPE *RETURN* FOR MORE

P [hex-address]

Example:

P 5C20

This is a new command which is similar to "M" in that they both start the execution of a program already in memory.

"P" differs from "M" in the default address. For "M" when no address is given, the Run address of the last loaded binary file is used. For "P", the address of the last P command is used.

! TYPE *RETURN* FOR MORE

P does not provide a prompt for parameter. This allows a very simple method of running a frequently used

its Run address. You can then execute the program by just typing:

P

and only a single line of the screen is used.

You can save the default address by the "H" Write-DOS-Files command.

*** END OF P ***

"Q" - COMMAND FILE

! TYPE *RETURN* FOR MORE

Q filespec

Example:

Q PRINTLST.CMD

This new command lets you execute a file of DOS commands. You can create a file of complex commands that can be later executed by a simple Q command. This will simplify your operations of commonly used commands.

COMMENT LINES

Comment lines, starting with ";" or " " (space), may occur anywhere in a command file. They are displayed on the screen, but cause no action.

! TYPE *RETURN* FOR MORE

"$" & "#" - CONTROL SECONDARY INPUT
(special commands)

"$" and "#" are special commands used to control the source of "secondary" input, when needed by Menu commands executed within a command file.

When used, these special commands must appear in the first column of a line. The rest of the line is not used by DOS-MOD, so may be comments.

By "secondary input" is meant any input required by a Menu command, which is not given on the command line. This would include parameters, if not given on the command line, and response to a query, such as required by the "H" and "I" commands.

! TYPE *RETURN* FOR MORE

For example:

```
I 2
```

to format disk drive-2 requires a subsequent "Y" confirmation to enable the operation.

If, in this example, only the "I" were used on the command line, then both the drive number "2" and the confirming "Y" would be required on subsequent lines as secondary input.

Whether this secondary input is taken from the next line or lines of the command file, or from the Screen (that is, from the user), is determined by which special command "$" or "#" was last issued.

! TYPE *RETURN* FOR MORE

The special command:

```
$
```

causes DOS to seek secondary input from the user, that is, the Screen Editor (E:).

Whereas the special command:

```
#
```

causes DOS to seek secondary input from the next line in the command file.

NOTE: To help remember which is which:
People like dollars ($), but computers like numbers (#).
So "$" gets secondary input from the user and "#", from the file.

! TYPE *RETURN* FOR MORE

REVIEW: For commands in the command file that require secondary input, the command file must specify by a prior "$" or "#" command whether the secondary input is to be obtained from from the user or from the next line of the command file.

For example, if the command "I 1" were in the command file, then a prior "$" would cause DOS to wait for the user to reply, while a prior "#" would cause DOS to take the next line of the command file as the answer.

Thus, when you create a command file, for each command line which requires secondary input, you can include the

let the user respond when the command file is executed.

! TYPE *RETURN* FOR MORE

EXAMPLE OF THE USE OF "$" AND "#"

To illustrate the use of "$" & "#", we execute a "C" Copy command with a /Q (Query) option which requires a response. "#" is used to cause the secondary input to be taken from the command file.

The following example will be executed in the next screen. It is shown here first, so you can see how the lines appear in this file.

```
#
C A,A.TMP/Q
Y
```

! TYPE *RETURN* FOR MORE

```
#
C A,A.TMP/Q
Y
```

! TYPE *RETURN* FOR MORE

Now we use "$" to show secondary input from the user. Now you can respond with a "Y" to delete the A.TMP file that was created.

This example appears in this file as:

```
$
D A.TMP
```

! TYPE *RETURN* FOR MORE

```
$
D A.TMP
```

! TYPE *RETURN* FOR MORE

RECURSION

A "Q" command file may itself contain "Q" commands, up to four levels deep (if enough open files are available).

Each level requires one open file. In addition, some commands within a command file may themselves require open files. For example, the "C" Copy command requires two open files.

For a discussion of how to specify the maximum number of open files, see TIP #7, in the TIPS tutorial.

I/O Channels 4-7 are used for Q files.

! TYPE *RETURN* FOR MORE

*** END OF Q ***

#
"R" - READ/STORE MEMORY
! TYPE *RETURN* FOR MORE

R hex-address

Example:

R 5C20

NOTE: This command is available only if you have the OSB-ROM. If you have the OSA-ROM (See the INTRO tutorial file), you may want to skip this R tutorial by *CTRL*-3, unless you want to see what you are missing!

This new command is used to examine and change bytes in memory. Hex notation is used throughout for addresses and byte values.

! TYPE *RETURN* FOR MORE

In response to the "R" command, DOS will display the given address, followed by the contents of the eight bytes of data at that location in memory. The data is displayed in both hex and ATASCII.

For example,

R 1F0F
X

will result in the following output:

! TYPE *RETURN* FOR MORE

R 1F0F
X

! TYPE *RETURN* FOR MORE

The prompt STORE is displayed to remind you that the Store-Memory mode is selected by "S".

When you are in the R command you have the choice of typing:

RETURN, to display the next 8 bytes,
S, to Store new values of the bytes,
Any DOS command, to execute, or

! TYPE *RETURN* FOR MORE

The first choice, *RETURN*, causes DOS-MOD to display the next eight bytes of memory.

For example, to display the eight bytes at 1F0F and the next three sets of eight bytes, you would type:

```
R 1F0F


X
```

This example is now executed:

```
! TYPE RETURN FOR MORE

R 1F0F


X

! TYPE RETURN FOR MORE
```

The second choice, "S", enables the Store-Memory mode.

In this mode, the memory address is repeated just under the line that displays the values of the 8 bytes. (The STORE prompt line is overwritten)

You can type in new hex values under the values displayed in the first line, using the standard line-editing facilities of OS. Zero to eight bytes can be changed on the line.

When you are satisfied with the new byte values, *RETURN* will cause the new values to be stored in memory.

The new values are again displayed so you can check to see that the results are what you wanted.

! TYPE *RETURN* FOR MORE

For example, to change the characters at 1F0F from "COPYRIGH" to "KOPIRITE", you would type:

```
R 1F0F
S
4B 4F 50 49 52 49 54 45
X
```

This example is now executed:

! TYPE *RETURN* FOR MORE

```
R 1F0F
S
4B 4F 50 49 52 49 54 45
X
```

! TYPE *RETURN* FOR MORE

Now we had better change the spelling back, or Atari might get angry!

```
R 1F0F
S
43 4F 50 59 52 49 47 48
X
```

! TYPE *RETURN* FOR MORE

The third choice is any legal DOS command. This is a quick way of exiting the "R" command, since the "R" command exit and the next command are accomplished at the same time on the same line.

The fourth choice, "X", causes an exit from the "R" command without executing another command.

After the first or second choice, you again have any of the four choices. You may continue in the "R" command until you exit by either the third or fourth choices.

! TYPE *RETURN* FOR MORE

You can try the "R" command if you'd like, using the same area of memory as in the above examples.

Try several *RETURN*s to get more of the message. If you try "S" to store changes, be sure you don't preserve the changes by later using the "H" Save-DOS-Files command.

REMEMBER TO TYPE "X" TO EXIT

! Type "R 1F0F" or *RETURN*

*** END OF R ***

BUGS

! TYPE *RETURN* FOR MORE

DOS-2.0 bugs fixed in DOS-MOD include:

1. MEM.SAV (N) does not work when:

   The 1st section of a binary file

but a subsequent section does.

A section of a binary file overlaps the DUP-area at both ends.

A binary file doesn't overlap the DUP-area, but it includes a byte in the same page as the DUP-area.

A program writes in the DUP-area.

MEM.SAV holds a program & a file not overlapping DUP is loaded.

! TYPE *RETURN* FOR MORE

2. RENAME (E) to an existing file creates a new file of that name without deleting the old file. This results in 2 files with the same name. This bug is especially troublesome when renaming a set of files using wildcards.

3. Scroll below the menu is faulty. It sometines rides up the screen and can hit the top and hamper the typing of commands. Even when the command section has ridden up the screen, it still is limited to only a few lines. The user really should have the whole screen to work with.

! TYPE *RETURN* FOR MORE

4. SAVE-FILE (K) does not accurately determine whether the area to be saved overlaps the DUP area and therefore will misbehave in some cases.

5. SAVE-FILE (K) aborts if the length of the area is greater than 8000 hex.

6. SAVE-FILE (K) does not give a report if an error, such as "disk full", occurs during the file-write process. [Bug fixed for OSB-ROM only.]

7. The address of the last byte of DUP is mistakenly set to the address of the byte beyond the last byte.

! TYPE *RETURN* FOR MORE

8. LOAD (L) uses a buffer in DUP, which could cause trouble if the program being loaded overlapped the area of that buffer.

9. In the Directory (A) command, if the second parameter (the list file) is inadvertently given as "D:", say instead of "P:", the file given as the filespec will be used as the list file

10. The RS-232 handler, which is loaded from the Atari-850 Interface Module, is destroyed on System Reset. [cf: Wilkinson, "COMPUTE", Dec 82, p.242]

! TYPE *RETURN* FOR MORE

11. When the directory is full (64 files), a Copy (C) command which would create a new file may cause DOS FMS to hang in a loop. The loop occurs when the Copy command has been preceded by a successful COPY.

*** END OF BUGS ***

TIPS & TRICKS

! TYPE *RETURN* FOR MORE

1. SCREEN SCROLL CONTROL

Remember that *CTRL*-1 will stop the screen from scrolling. (A second *CTRL*-1 will restart the screen.)

2. COPY TO THE SCREEN

You can use the screen (E:) to look at a text file. Use the "C" command to copy the file to (E:).

For example, try copying one of these tutorial files to the screen. The "B" file is a short one.

! Type "C B,E:" or *RETURN*

! TYPE *RETURN* FOR MORE

3. COPY FROM THE SCREEN

You can copy from the Screen Editor (E:) to create a text file. Simply Copy (E:) to the desired disk file.

After the "C" command is executed, DOS will wait for you to type in lines to the Screen Editor. You may edit each line as you go along, using all of the Screen Editor's facilities.

When you are satisfied with each line, a *RETURN* will transmit it to the file. When you have typed in all the desired lines, type *CTRL*-3 (End-of-File).

For example, try creating a test file of two or three lines (remember, DOS

! Type "C E:,TEST.TXT" or *RETURN*

Now you can look at your file:

! Type "C TEST.TXT,E:" or *RETURN*

! TYPE *RETURN* FOR MORE

4. EDITING A TEXT FILE ON THE SCREEN

You can edit small text files -- up to about 20 lines -- using the Screen Editor and the "C" Copy command:

Copy the file to the screen (E:).
Execute the command to copy the screen (E:) back to the file.
DOS waits for you to transmit lines to the edited file.
Edit each line on the screen by moving the cursor to the line, using the cursor commands.
Do not use RETURN to move cursor.
Transmit each edited line to the file, in desired order, with *RETURN*.
When edit is complete, move cursor to bottom of screen and type *CTRL*-3.

For example:

! Try typing "C TEST.TXT,E:" or *RETURN*

Your TEST.TXT file should be on the screen.

After the next command, you will be asked if it's OK to delete your old file. Type "Y". Then use the above technique to edit and enter lines in the new file.

BE SURE TO MOVE THE CURSOR TO THE BOTTOM OF THE SCREEN, BEFORE YOU TYPE THE CTRL-3 (END-OF-FILE)!!!

! Now type "C E:,TEST.TXT" or *RETURN*

Now look at your edited file.

! Now type "C TEST.TXT,E:" or *RETURN*

! TYPE *RETURN* FOR MORE

5. WILDCARDS FOR A SINGLE-FILE COPY

A special use of wildcards in DOS-MOD permits a simpler copy or rename of a single file.

A wildcard in the to-file is replaced by the corresponding filename part of the from-file.

This method is used when copying or

name, but different extension -- or the same extension, but a different name. Examples:

```
C ANYFILE.TXT,*.OLD
E ANYFILE.TXT,*.OLD
C ANYFILE.TXT,OLDFILE.*
```

! TYPE *RETURN* FOR MORE

6. HOW TO GET THE NUMBER OF FREE FILES

There is a limit of 64 files to a disk directory. When you are nearing this limit, it would be nice to know, without counting, how many more files you can create.

In DOS-MOD, there is now a simple way to find out how many "free" files are available.

After an "A" command is executed, the byte at 74 ($4A) will hold this value. You can read it with the new "R" command (the number will be in hex), or with the BASIC PEEK command (the number will be in decimal).

! TYPE *RETURN* FOR MORE

7. DISK DRIVE CONFIGURATION

The Atari DOS II Manual, Appendix G, describes how to configure your disk drives. There is a bit for each disk drive unit in the byte at 1802 ($70A).

When DOS is initialized on power-up or on *SYSTEM-RESET*, one 128-byte buffer is allocated for each drive unit for which a bit exists in the drive byte. These buffers are located between the areas occupied by DOS.SYS and DUP.SYS. (See the picture in the "N" tutorial.)

In addition to the buffers allocated for the disk drives, one buffer is required for each concurrently open file. The number of buffers available for open files is set by the byte at 1801 ($709).

! TYPE *RETURN* FOR MORE

SPACE AVAILABLE FOR BUFFERS

There is space for six 128-byte buffers in the buffer area.

To stay within this buffer space, you must limit to six the total number of drives plus the number of open files.

However, if you have a need for more

this formula would allow, you might experiment with setting the limit higher. But you should consider the following factors if doing so.

! TYPE *RETURN* FOR MORE

EXCEEDING SIX BUFFERS

If the total number of buffers does exceed six, the excess buffers will overlay the DUP-area and, when used, will destroy the DUP Menu program.

USE OF OPEN FILES BY USER PROGRAM

If a user program opens files which then use the excess buffers, the damaged DUP Menu program will be restored from DUP.SYS file by DOS-MOD when you return to the DUP Menu. The only loss is the few seconds it takes to read DUP.SYS.

! TYPE *RETURN* FOR MORE

USE OF OPEN FILES BY DUP MENU

If a DUP Menu program opens files which use the excess buffers, while you are executing Menu commands, then expect a disaster! Avoid this!

! TYPE *RETURN* FOR MORE

OPEN-FILE USE BY DOS-2.0 and DOS-MOD

DOS-2.0 uses two open files, so up to four disk drives can be configured without conflict between buffer use and the DUP Menu program.

DOS-MOD uses two open files, plus one open file for each level of command files used under the new Q command.

For example, if you call a "Q" command file, which in turn calls another command file, two open files are needed for the command files, and a total of four open files are required altogether. In this case, you can safely configure up to 2 disk drives.

! TYPE *RETURN* FOR MORE

CONFIGURATION BYTES, AS DELIVERED

You will find that the Atari DOS 2.0S, as delivered, is configured for disk drives 1 & 2 and for 3 open files.

DOS-MOD, as delivered, is configured for drives 1 & 2 and for 4 open files.

the "DEMO" tutorial.

HOW TO CHANGE CONFIGURATION

You can change the disk drive bytes at locations 1801 ($709) and 1802 ($70A) by using the new "R" command or by the POKE command in BASIC.

! TYPE *RETURN* FOR MORE

8. TUTORIAL FILE LISTING

If you have a printer, you can get a listing of the tutorial files, using the new "Q" Command File command. A command file to accomplish this is included on the delivered disk.

To get this printer listing, you would type "Q TUTORIAL.CMD".

To see what this command file looks like (be ready with *CTRL*-1 to hold the screen):

! Type "C TUTORIAL.CMD,E:" or *RETURN*

! TYPE *RETURN* FOR MORE

If you would like to get the printed listing now, you can execute the "TUTORIAL.CMD" file. (It's quite long, so you may want to get it later.)

NOTE: These tutorial files contain special control characters that may cause peculiar effects on some printers.

To get the printed listing now:

! Type "Q TUTORIAL.CMD" or *RETURN*

*** END OF TIPS ***